5
Implementação de Suporte à Geração Automática de Aplicações para o Ginga-NCL

O objetivo deste capítulo é apresentar a implementação do suporte à geração automática de aplicações para o módulo **SAGGA1**, que foi descrito anteriormente. A partir dessa implementação, há uma base de conhecimento a ser reaproveitada em outras aplicações que se recriam dinamicamente, como mostra a Seção 4.2. O trabalho de implementação realizado nesta dissertação pode ser dividido em duas partes.

A primeira parte tem foco nas especificações dos objetos que compõem a arquitetura da aplicação **SAGGA1**, apresentada na Figura 4.5. Esses objetos dão suporte à geração de conteúdo dinâmico em aplicações que se recriam dinamicamente.

A segunda parte, por sua vez, consiste na especificação de uma ferramenta gráfica que auxilia na autoria de documentos **SAGGA**. A criação dessa ferramenta se justifica no fato de que o método de autoria orientado a *templates* de (Soares Neto, 2010), que foi apresentado na Seção 2.2 desta dissertação, prevê a existência de um ambiente de autoria para documentos. Segundo esse método, tal ambiente deve prover uma linguagem para instanciação de *templates*, ou seja, um meio para autores de documentos especificarem como querem instanciar *templates*. Em suma, a instanciação dos *templates* corresponde à autoria do documento pronto para ser exibido.

A ferramenta apresentada neste capítulo se destina a prover uma linguagem gráfica para instanciação de *templates* dos módulos **SAGGA**. Por meio dessa ferramenta, não somente um programador experiente é capaz de criar aplicações **SAGGA**, mas também outros profissionais que não tenham esse perfil, pelo fato de que a ferramenta cria uma abstração gráfica para especificar documentos.

5.1. Especificação da aplicação SAGGA1

A estrutura da aplicação **SAGGA1** consiste em um documento composto por um objeto de mídia NCLua que possui a função de gerador de conteúdo. O gerador de conteúdo é responsável por manipular, em tempo de apresentação, o conteúdo que está sendo exibido no momento. Um exemplo desse documento é ilustrado na Figura 4.2, apresentada no Capítulo 4. Por meio de eventos, o agente de usuário do Ginga (formatador) permite que haja comunicação entre documentos NCL e outros objetos de mídia. Esse meio de comunicação é explorado na especificação da comunicação entre o documento e o objeto gerador de conteúdo da aplicação **SAGGA1**.

Voltando à Figura 4.2, nela há uma listagem com um exemplo de documento inicial de **SAGGA1**. Nessa figura, o objeto NCLua gerador de conteúdo (*media id="generator"* na linha 14) possui uma série de propriedades. Eventos de atribuição a essas propriedades são tratados por esse objeto gerador por meio de uma função de tratamento de eventos, como ilustra a função *handler* na Figura 5.1. Os eventos de atribuição sobre cada propriedade preparam o gerador para receber uma operação de recriação do conteúdo, que acontece quando um evento de atribuição na propriedade "data" é disparado pelo documento.

```
1.  local function handler(evt)
2.      if evt['class'] == 'ncl' and  evt['type'] == 'attribution' and evt['action'] == 'start'  then
3.        if evt['name'] == 'builder' then
4.              set_builder(evt['value'])
5.        elseif evt['name'] == 'template_processor' then
6.              set_template_processor(evt['value'])
7.        elseif evt['name'] == 'template' then
8.              set_template(evt['value'])
9.        elseif evt['name'] == 'data' then
10.             set_data(evt['value'])
11.       end
12.     end
13.     evt['action'] = 'stop'
14.    event.post(evt)
15. end
```

Figura 5.1 - Função handler do NCLua gerador de conteúdo de SAGGA1

A propriedade “builder” – tratada nas linhas 3 e 4 da Figura 5.1 – contém a informação de qual gerador de documento deve ser utilizado. No caso da aplicação **SAGGA1**, essa variável é configurada com o nome do módulo NCLua que gera documentos de preenchimento de **SAGGA1**. A possibilidade de configurar “builder” permite que esse objeto gerador seja reutilizado para outras aplicações que se recriam dinamicamente.

A propriedade “template_processor” – tratada nas linhas 5 e 6 da Figura 5.1 – contém a informação de qual processador de *templates* deve ser utilizado. A possibilidade de configurar “template_processor” permite que sejam utilizados processadores de *templates* para diferentes tipos de linguagens de *templates*. Por exemplo, poderia ser utilizado um processador de *templates* para a linguagem TAL (do inglês, *Template Authoring Language*) (Soares Neto, 2010), se o *template* utilizado na aplicação fosse especificado em TAL. Nesta dissertação, os *templates* são especificados por meio de uma linguagem na qual documentos NCL possuem código Lua embutido com as restrições sobre o *template* e o processador de *templates* utilizado é um objeto NCLua que faz um *parser* dessa linguagem.

5.1.1. Template e documento de preenchimento

A propriedade “template” – tratada nas linhas 7 e 8 da Figura 5.1 – contém a informação de qual é o *template* utilizado para construir a aplicação. Por exemplo, para criar uma aplicação com *template* **SAGGA1**, a propriedade “template” recebe o documento que contém esse *template*, por meio do qual o processador de *templates* irá extrair informações para construir a aplicação.

Ainda no exemplo da Figura 4.2, as linhas 36 e 37 da listagem correspondem à inicialização de um evento de atribuição para a propriedade “template” com o valor “template_sagga1.ncl”. O documento referente a “template_sagga1.ncl” representa o *template* de composição para aplicações **SAGGA1** e se encontra no Anexo II desta dissertação. A linguagem utilizada neste *template* é aquela mencionada anteriormente, com documentos NCL embutindo código Lua.

A característica dos documentos **SAGGA1** é possuir um vídeo principal, exibir uma legenda e conter menus de vídeos relacionados oriundos dos *sites*. É possível estabelecer as seguintes restrições de cardinalidade sobre os documentos **SAGGA1**:

I. Há um e somente um vídeo principal no documento em qualquer momento;
II. Há um e somente um arquivo de legendas sendo exibido em qualquer momento;
III. Há no mínimo 1 vídeo relacionado no menu de cada *site*.

A Figura 5.2 apresenta uma listagem com um trecho do código do *template* que restringe a cardinalidade do vídeo principal. As restrições de cardinalidade para as legendas e vídeos relacionados são especificadas de modo semelhante e foram omitidas desta explicação. O laço de iteração *for* (linha 4) só é executado se a cardinalidade do conjunto de vídeos principais no documento de preenchimento for igual a 1 (teste *components.mainVideo* == 1 na linha 1 da Figura 5.2). Se o laço é executado, é feita uma iteração pela tabela *components.mainVideo*, onde a variável *i* corresponde ao índice durante a iteração e a variável *v* corresponde ao *i-ésimo* elemento na tabela *components.mainVideo*. A tabela *components.mainVideo*, como é visto adiante, faz parte do documento de preenchimento. O laço adiciona no documento a porta com *id* = *p_mainVideo_1* (linha 4), e a mídia com *id* = *mainVideo_1* (linha 5), cujo *src* (linha 6) é a lacuna preenchida pelo elemento *v* do documento de preenchimento.

```
1.  [! if components.mainVideo ~= nil and #components.mainVideo == 1 then
2.      for i,v in ipairs(components.mainVideo) do
3.  !]
4.  <port id="p_mainVideo_[!=i!]" component="mainVideo_[!=i!]"/>
5.  <media id="mainVideo_[!=i!]" type="video/mpeg"
6.          src="[!=v.src!]" descriptor="dt_mainVideo_[!=i!]"/>
7.  [! end end!]
```

Figura 5.2 – Trecho do *template* SAGGA1 para cardinalidade do vídeo principal

Os *templates*, como tem sido afirmado ao longo deste trabalho, constituem-se em um importante meio de reuso de especificações. O *template* de **SAGGA1**

contém alguns comportamentos recorrentes que foram identificados para as aplicações conforme tal *template*. A Figura 5.3 possui uma listagem com um exemplo de comportamento recorrente em **SAGGA1**. Ele expressa que seja gerado um *link* (linha 10 a 21) que, ao iniciar o objeto NCLua exibidor de legendas SRT (condição na linha 11), seja atribuída a cor amarela (*yellow*) à sua propriedade *color* pela ação nas linhas 12 a 14, o caminho (*path*) contido em *v.path* à sua propriedade *path* pela ação nas linhas 15 a 17 (lembrando que *v* é o elemento do documento de preenchimento obtido pela iteração de *components.srtLua*) e a fonte vera à sua propriedade *font* pela ação nas linhas 18 a 20. Esse comportamento só precisa ser definido no *template*, e será gerado para todos os documentos dessa família pelo processador de *templates*.

```
1.  [! if components.srtLua ~= nil and #components.srtLua == 1 then
2.      for i,v in ipairs(components.srtLua) do
3.  !]
4.  <port id="p_luaSrt_[!=i!]" component="srtLua_[!=i!]"/>
5.  <media id="srtLua_[!=i!]" src="./luadir/mediaplayer_srt.lua" descriptor="dt_srtLua_[!=i!]">
6.      <property name="path"/>
7.      <property name="color"/>
8.      <property name="font"/>
9.  </media>
10. <link xconnector="saggaBase#onBeginSet">
11.     <bind role="onBegin" component="srtLua_[!=i!]"/>
12.     <bind role="set" component="srtLua_[!=i!]" interface="color">
13.         <bindParam name="val" value="yellow"/>
14.     </bind>
15.     <bind role="set" component="srtLua_[!=i!]" interface="path">
16.         <bindParam name="val" value="[!=v.path!]"/>
17.     </bind>
18.     <bind role="set" component="srtLua_[!=i!]" interface="font">
19.         <bindParam name="val" value="vera"/>
20.     </bind>
21. </link>
  …
22.  [!end end!]
```

Figura 5.3 – Exemplo de comportamento recorrente no template SAGGA1

O processador de *templates* utilizado nesta dissertação recebe documentos de preenchimento como tabelas Lua. As lacunas são deixadas nos *templates* e o

gerador de documentos de preenchimento precisa gerar tabelas com valores que preencham essas lacunas. A Figura 5.4 ilustra um exemplo de possível documento de preenchimento representado por tabelas Lua gerado pelo gerador de conteúdo dinâmico para **SAGGA1**. A tabela *components* armazena as tabelas com as outras classes de objetos de mídia. A tabela *mainVideo* possui um elemento, cujo atributo *src* é o preenchimento da lacuna da classe dos vídeos principais, como a que aparece na Figura 5.2; A tabela *srtLua* possui um elemento, cujo atributo *path* é o preenchimento da lacuna da classe das legendas; por fim, a tabela *related* possui vários elementos, omitidos por questões didáticas sem interferência no entendimento do texto.

```
1.  components = {
2.      mainVideo =
3.      {
4.          {
5.              src="http://www.overmundo.com.br/download_banco/arte-sucata-reproducao-
    sonora-mecanica"
6.          }
7.      },
8.      srtLua =
9.      {
10.         {
11.             path="some_subtitle.srt"
12.         }
13.     },
14.     related =
15.     {
16.         ...
17.     }
18. }
```

Figura 5.4 – Exemplo de documento de preenchimento de **SAGGA1**

5.1.2. Recriando documentos SAGGA1 dinamicamente

A atribuição de valor à propriedade "data" – tratada nas linhas 9 e 10 da Figura 5.1 – invoca o método *setData*. Esse método dispara a recriação da aplicação, realizada conforme os parâmetros configurados no gerador de

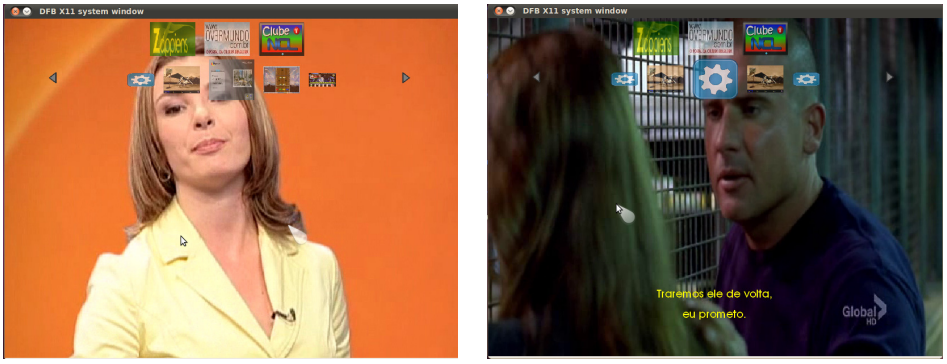
conteúdo. Por meio de atribuições à "data", o documento pode ser recriado quando necessário.

A arquitetura ilustrada na Figura 4.6 contém um objeto "Lua Controlador". Esse objeto controla a recriação da aplicação. Ele possui um método *getContext* que recebe o valor de "data". O método *getContext* tem como resultado um objeto de composição (nó *context*) que será adicionado pelo gerador por comando de edição no documento após a remoção da composição anterior.

Para construir o contexto que será devolvido ao gerador, inicialmente "Lua Controlador" invoca *getDoc(data)* do gerador de documentos de preenchimento (representado por "FillDocGen" na Figura 4.6) que é identificado pela propriedade "builder". O gerador de documentos de preenchimento faz o *parser* de "data", e constrói o documento conforme os dados recebidos. Por exemplo, em **SAGGA1** a aplicação é recriada quando um vídeo relacionado é selecionado. O valor de "data" deve conter informações sobre qual vídeo foi selecionado, a fim de que o gerador de documentos de preenchimento construa o menu de vídeos relacionados ao vídeo selecionado.

Após obter o documento de preenchimento, "Lua Controlador" invoca *getContext(doc, template)* do Processador de *Templates* (Figura 4.6) que é identificado pela propriedade "template_processor" do gerador de conteúdo.

A Figura 5.5 apresenta capturas de tela da aplicação **SAGGA1** sendo executada em dois momentos sequenciais. A Figura 5.5 (a) demonstra que um vídeo está sendo exibido e o usuário está navegando em vídeos relacionados. Ao selecionar um dos vídeos, o vídeo é trocado e um novo menu de vídeos relacionados é exibido, como representa a Figura 5.5 (b). Percebe-se que a ação de escolher o vídeo corresponde à autoria do próximo momento da aplicação.



(a) (b)

Figura 5.5 - Aplicação SAGGA sendo exibida em momentos (a) e (b) seqüenciais

5.1.3. Ambiente de execução do gerador de conteúdo

No ambiente da *Web*, uma decisão de projeto recorrente no processo da criação de páginas dinâmicas se refere ao ambiente onde será executado o código imperativo que realiza a geração dinâmica do conteúdo. Se a geração é simples e pode ser feita no ambiente do cliente, a página comumente é estruturada com objetos JavaScript, *applets* Java, como foi mencionado no Capítulo 2. Por outro lado, a página pode ser estruturada de forma que sua geração dinâmica seja executada no servidor, com *scripts* PHP, ASP etc.

Em relação ao contexto de TV Digital, onde estão inseridas as aplicações abordadas neste trabalho, a diferença reside na forma de entrega utilizada pelo provedor de conteúdo para o cliente. Na *Web*, o usuário solicita uma página a um servidor *Web* e a entrega da página e seus elementos é feita através de um canal de HTTP (unicast). No entanto, a entrega do documento hipermídia em um ambiente de execução de TV Digital utiliza canais de comunicação *broadcast* (transmissão por via terrestre, cabo, satélite etc.).

Por utilizar comunicação *unicast*, a *Web* torna possível que a geração dinâmica do conteúdo possa ser feita tanto no cliente quanto no servidor. Por sua vez, se um receptor de TV Digital só tiver acesso a uma rede de recepção *broadcast*, não há como o conteúdo ser gerado dinamicamente no ambiente do transmissor da aplicação, pois o receptor não possui canal de retorno, impossibilitando de enviar informações para o servidor de aplicações na emissora de TV Digital.

Além da geração dinâmica no cliente na TV Digital, é possível elaborar uma solução utilizando servidores *proxy*. Essa solução demanda uma infra-estrutura de acesso à rede onde se encontra o servidor *proxy*. Uma aplicação NCL poderia conter objetos NCLua, que por sua vez acessariam outros objetos localizados remotamente nesse *proxy*.

A solução local para o gerador da Figura 4.6 é implementar todos os objetos NCLua localmente. Essa solução é mais simples, porém tem a desvantagem de poder sobrecarregar a máquina do cliente dependendo dos recursos necessários

para a geração dinâmica, além de impossibilitar alguns tipos de manipulações sobre dados que o cliente não possui acesso.

A solução por p*roxy* para a geração de conteúdo dessa arquitetura consiste em implementar o objeto "Lua Gerador" no ambiente do cliente, e os objetos "Lua Controlador", "FillDocGen" e "Processador de *Templates*" em *proxies*. A Figura 5.6 apresenta a configuração da mesma visão estrutural da Figura 4.6, fazendo a separação entre os objetos que ficam no servidor proxy e no cliente. A abordagem com servidor *proxy* possui as vantagens de permitir a geração de conteúdo mesmo quando clientes dispõem de poucos recursos, isolar dos clientes os dados para geração de conteúdo etc. No entanto, como foi mencionado, essa abordagem requer uma infra-estrutura de acesso ao servidor *proxy*.

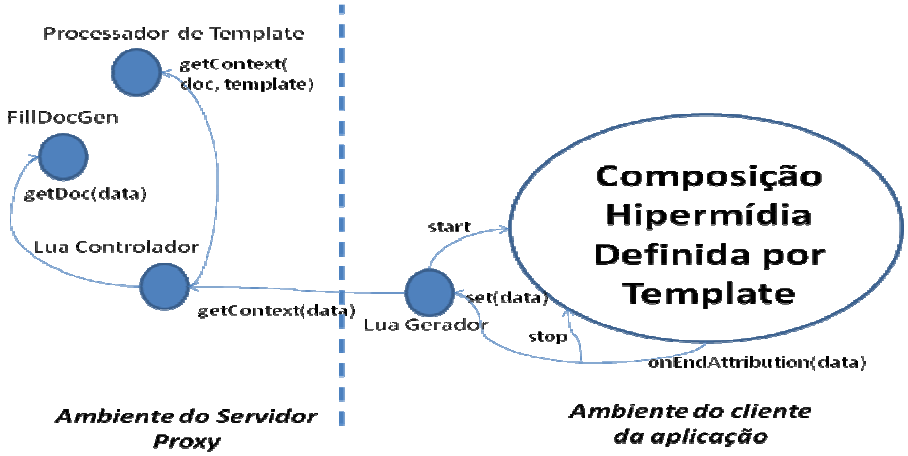


Figura 5.6. Arquitetura para aplicações que se recriam dinamicamente com servidor *proxy*

O objeto "Lua Controlador" é muito útil nessa arquitetura porque, se ele não estivesse presente, "Lua Gerador" deveria ter interface direta com "FillDocGen" e "Processador de *Templates*". Isso resultaria em um "Lua Gerador" específico por aplicação. Com a presença do "Lua Controlador", o objeto "Lua Gerador" faz requisições *get(data)* ao "Lua Controlador". O parâmetro *data* das requisições *get* deve conter a informação de qual combinação gerador de documento de preenchimento, *template* e processador de *templates* deve ser utilizada para a geração. Dessa forma, o "Lua Controlador" se encarrega de obter o documento de preenchimento diretamente com o "FillDocGen" e a composição hipermídia com

o "Processador de *Templates*". Essa estrutura faz com que o "Lua Gerador" seja genérico para as aplicações dessa arquitetura.

5.2. Ferramenta para autoria de documentos

A ferramenta apresentada a seguir provê aos usuários uma abstração gráfica para especificar documentos de preenchimento para aplicações **SAGGA**. Apesar de esta dissertação ter foco no módulo **SAGGA1**, considerando-se que os outros módulos do projeto **SAGGA** são também aplicações que se recriam dinamicamente, a ideia é que essa ferramenta seja estruturada para ser extensível a fim de permitir o acoplamento futuro de outros módulos.

A implementação da ferramenta para **SAGGA** foi feita utilizando *Qt* (Blanchette, 2006). *Qt* é um *framework* para desenvolvimento em C++ que permite, entre outras coisas, desenvolver *softwares* aplicativos com interface gráfica para usuário, chamada de GUI (do inglês, *Graphical User Interface*).

Um dos pontos analisados na decisão de utilizar *Qt* nesta implementação diz respeito às facilidades que esse *framework* disponibiliza para desenvolver aplicações com GUI. Além dessa vantagem, pretende-se que a ferramenta **SAGGA** seja adaptada como um *plug-in* para *Composer* (Lima, 2010).

Composer é uma ferramenta que tem o objetivo de permitir a autoria de documentos NCL sob visões que se adequem aos diferentes perfis de autores. Para um usuário com perfil de programador, que se sente à vontade em trabalhar com código-fonte, a visão textual do documento NCL é útil. No entanto, um usuário que não se sinta à vontade com a visão textual pode ser contemplado com visões gráficas, como as visões de leiaute, estrutural etc.

Pelo fato de o *Composer* ser capaz de receber implementações de visões por meio de *plug-ins*, é possível que a ferramenta **SAGGA** seja adicionada ao *Composer* como uma visão sobre um tipo particular de documento NCL. A necessidade de implementação baseada no *Qt* leva em consideração que essa é a API gráfica que o *Composer* requer para a especificação dos *plug-ins*.

5.2.1. Diagrama de classes

A Figura 5.7 apresenta um diagrama de classes simplificado da ferramenta **SAGGA**. *MainWindow* é a classe que herda de *QMainWindow*, uma classe de *Qt* que implementa o controle de uma janela principal. O objeto de *MainWindow*, portanto, contém a lógica da janela principal da aplicação. Ele possui métodos *slot* para criar (*newApp*), abrir (*openApp*), salvar (*saveApp*), executar (*runApp*) e fechar (*closeApp*), que são chamados quando as suas respectivas ações disparam um *signal* para executar a tarefa.

MainWindow é composta por três objetos de classes de visualização sobre o modelo de documento **SAGGA**. As classes de visualização implementam as visões de leiaute, *outline* e propriedades, descritas na próxima seção. A ferramenta foi estruturada como uma espécie de implementação do padrão *Model-View-Controller* (MVC) (Gamma, 1995). A diferença é que não há uma classe de controle específica. O controle do sincronismo entre a visão e o modelo é feito pela própria *MainWindow* e pelas visões.

A visão de documento (*DocView*) possui um modelo (*DocModel*) associado a ela. Quando o objeto *MainWindow* instancia as visões, ele passa pelo parâmetro do construtor o objeto de modelo, assim as mudanças no modelo podem refletir nas visões. O objeto de modelo armazena as propriedades do modelo em processo de autoria. Assim, quando o usuário dispara a ação de salvar, o documento é atualizado no respectivo arquivo.

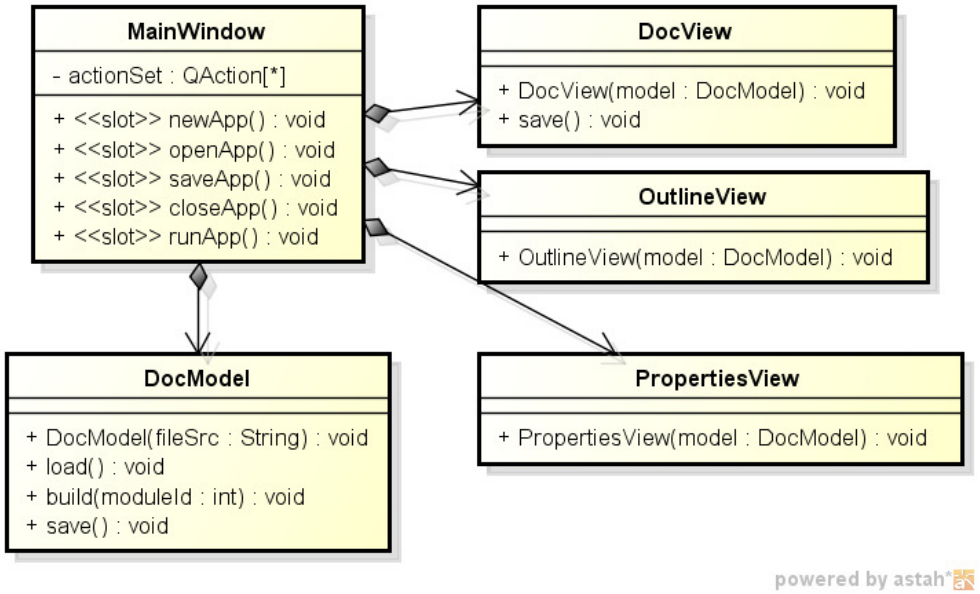


Figura 5.7. Diagrama de classes simplificado da ferramenta SAGGA

5.2.2. Funcionalidades

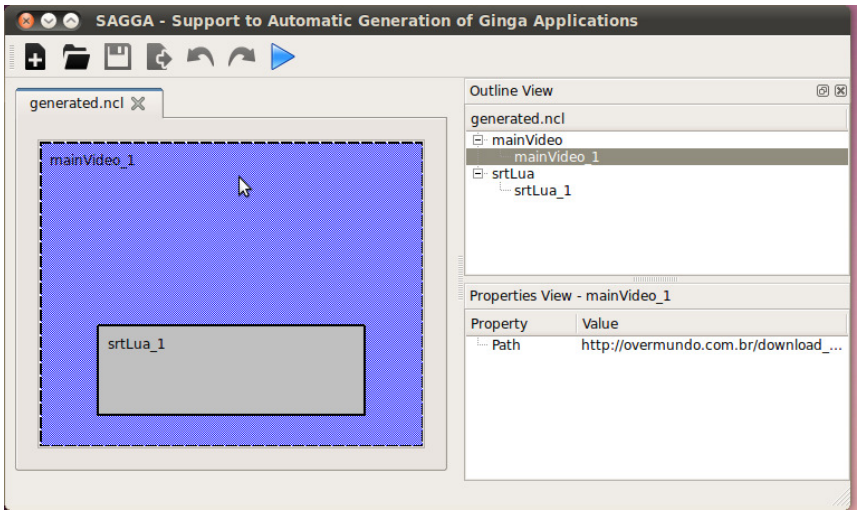
A Figura 5.8 apresenta a janela principal da ferramenta **SAGGA**. Para permitir a visualização do documento que está sendo criado ou aberto, a ferramenta possui visões de leiaute, *outline* e propriedades dos objetos de mídia; a ação de salvar realiza a persistência das modificações feitas no documento; por fim, a ação de executar invoca o agente de usuário Ginga para exibir o documento.

A visão de leiaute apresenta a disposição dos objetos no espaço da aplicação. Cada objeto aparece associado ao seu identificador. Na Figura 5.8, o arquivo "generated.ncl" possui os objetos "mainVideo_1" e "srtLua_1". A área dos objetos pode ser selecionada, redimensionada e movida. Na Figura 5.8 (a), o objeto "mainVideo_1" aparece selecionado, pois a linha que desenha a sua área está pontilhada e o preenchimento da área está com cor azul. Já na Figura 5.8 (b), a seleção muda para o objeto "srtLua_1", que foi movido e redimensionado.

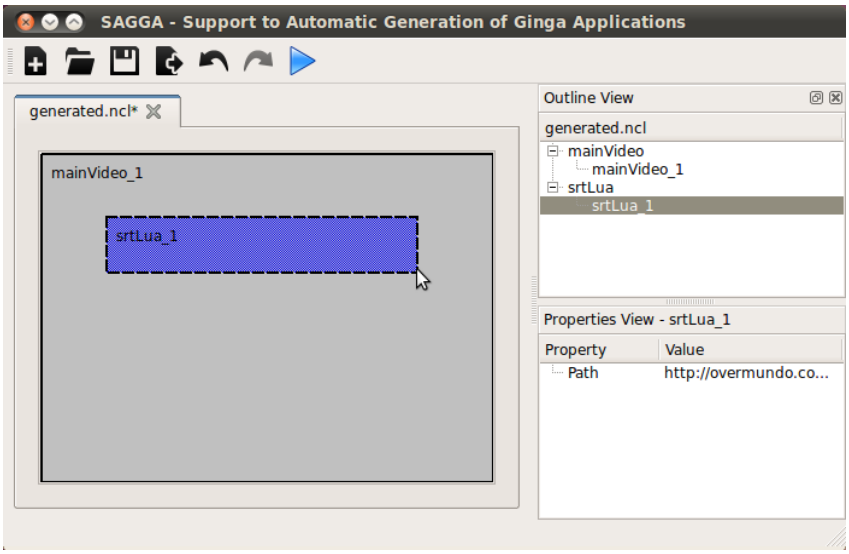
A visão de *outline* apresenta um resumo dos objetos de mídia que aparecem no documento, associados às suas respectivas classes de objetos do *template*. Por exemplo, na Figura 5.8 os objetos "mainVideo_1" e "srtLua_1" aparecem na visão de *outline* associados à classe "mainVideo" (vídeo principal) e "srtLua"

(legenda SRT), respectivamente, do *template* de **SAGGA1**. A visão de *outline* sincroniza com a visão de leiaute qual objeto está selecionado.

A visão de propriedades apresenta as propriedades do objeto de mídia selecionado. Por exemplo, na Figura 5.8 (a), a propriedade "Path" do objeto "mainVideo_1" – que está selecionado – é exibida na visão de propriedades.



(a)



(b)

Figura 5.8 – Janela principal de SAGGA em dois momentos (a) e (b)

5.2.3. Etapas para a geração de um documento SAGGA

A ação de criar um novo documento dispara o início de um objeto assistente (*Wizard*) para criação de novos documentos. A Figura 5.9 apresenta as quatro etapas na construção do novo documento. A primeira etapa refere-se às boas-vindas ao assistente (Figura 5.9 (a)). A segunda etapa, representada pela Figura 5.9 (b) diz respeito à escolha de qual tipo de aplicação **SAGGA** o usuário deseja criar. O assistente está preparado para receber opções de um dos três tipos de aplicações **SAGGA** (no entanto, somente o módulo **SAGGA1** foi implementado nesta dissertação). A terceira etapa (Figura 5.9 (c)) permite ao usuário escolher o diretório e o nome do arquivo no qual a aplicação estará armazenada. Por fim, a quarta etapa apresenta a confirmação das opções escolhidas pelo usuário (Figura 5.9 (d)).

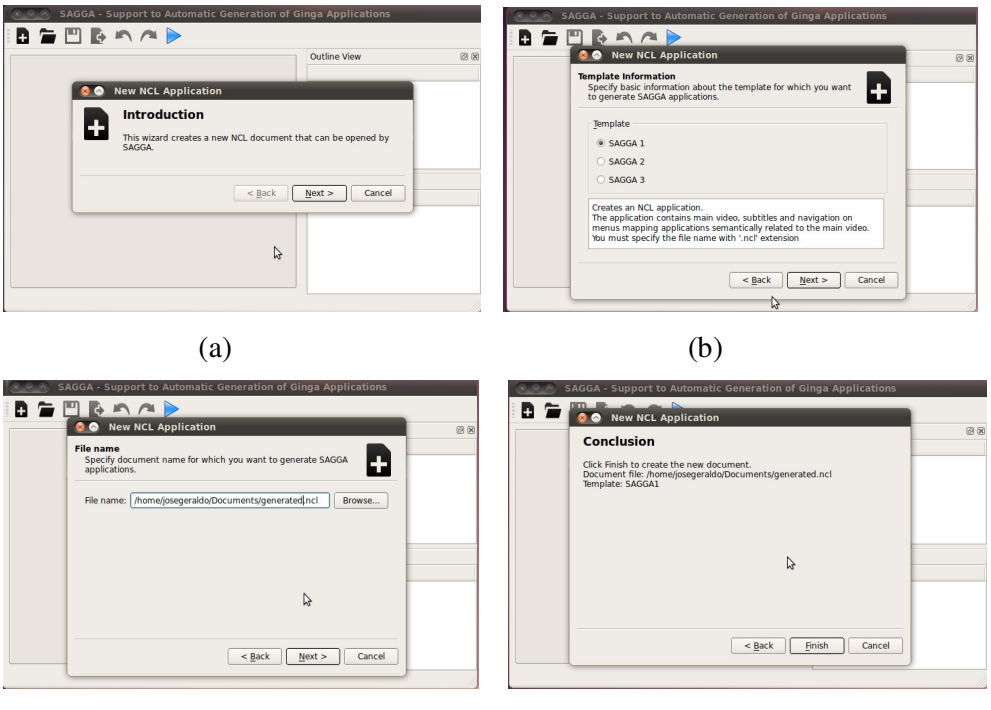


Figura 5.9 - Assistente para novo documento SAGGA nas etapas referentes a: (a) Introdução; (b) Escolha do tipo de aplicação (*template*); (c) Escolha do nome do arquivo NCL; (d) Confirmação da criação do documento

Quando o usuário escolhe um dos tipos de aplicação **SAGGA**, como está representado na Figura 5.9 (b), ele opta por um *template* de aplicação que deseja

construir. Para que a ferramenta seja capaz de abrir o documento com o *template* adequado, essa opção de *template* é salva como um elemento *meta* no arquivo NCL da aplicação. A Figura 5.10 representa um elemento *meta* que armazena de forma bastante simples a informação de que a aplicação corresponde ao módulo **SAGGA1**.

```
<meta name="template" content="sagga1"/>
```

Figura 5.10 - Metadado em arquivo NCL informando o tipo de *template* de uma aplicação

Ao final da criação de um novo documento, a ferramenta cria um documento com configurações *default* para o *template* de aplicação escolhido. Para o **SAGGA1**, por exemplo, a configuração *default* estabelecida possui os objetos de mídia na mesma disposição de leiaute que aparece na Figura 5.8 (a), com a propriedade "Path" (corresponde ao atributo *src* dos objetos de mídia NCL) vazio.

A partir do documento inicial criado pela ferramenta, o usuário realiza as modificações que deseja aplicar a esse documento. Ele pode configurar as propriedades dos objetos de mídia (como, por exemplo, "Path" do vídeo principal e das legendas), redimensionar a região onde os objetos de mídia serão exibidos etc. Um documento modificado pelo usuário aparece ilustrado na Figura 5.8 (b).

As modificações nas lacunas do *template* refletem no documento NCL propriamente dito alterando o valor do *bindParam* que é passado no evento de atribuição do *link_init* (vide Figura 4.2 entre as linhas 25 e 45) para a propriedade "data" (esse evento se encontra entre as linhas 42 e 45 da Figura 4.2). Com o *bindParam* do evento de *set(data)* configurado, a aplicação está pronta para ser exibida, com um aspecto semelhante à exibição da Figura 5.5